Solving Kinetic Equations on GPUs I: Model Kinetic Equations

A. Frezzotti, G. P. Ghiroldi, L. Gibelli *

Politecnico di Milano, Dipartimento di Matematica, Piazza Leonardo da Vinci 32, 20133 Milano, Italy

Abstract

We present an algorithm specifically tailored for solving kinetic equations onto GPUs. The efficiency of the algorithm is demonstrated by solving the one-dimensional shock wave structure problem and a two-dimensional low Mach number driven cavity flow. Computational results show that it is possible to cut down the computing time of the sequential codes of two order of magnitudes. The algorithm can easily be extended to three-dimensional flows and more general collision models.

Key words: Boltzmann equation, Deterministic methods, Parallel algorithms, Graphics Processing Units, CUDA™ programming model
PACS: 02.70.Bf, 47.45.Ab, 51.10.+y

1 Introduction

A recent trend emerging in computational physics stems from the availability of low cost general purpose graphics processing units (GPUs). GPUs have been used to accelerate CPU critical applications such as simulations of hypersonic flows [1], magnetized plasma [2] and molecular dynamics [3]. However, no applications to kinetic theory of gases seem to have been considered yet. Kinetic theory of gases deals with non-equilibrium gas flows which are met in several different physical situations ranging from the re-entry of spacecraft in upper planetary atmospheres to fluid-structure interaction in small-scale

* Corresponding author.
Email addresses: `aldo.frezzotti@polimi.it` (A. Frezzotti), `gian.ghiroldi@mail.polimi.it` (G. P. Ghiroldi), `livio.gibelli@polimi.it` (L. Gibelli).

devices [4,5]. The dynamics of dilute (or rarefied) gas flows is governed by the Boltzmann equation [6] which takes the form

$$\frac{\partial f}{\partial t} + \boldsymbol{v} \circ \nabla_{\boldsymbol{x}} f + \frac{1}{m} \nabla_{\boldsymbol{v}} \circ (\boldsymbol{F} f) = \mathcal{C}(f, f) \tag{1}$$

$$\begin{aligned} \mathcal{C}(f,f) = \int & [f(\boldsymbol{x}, \boldsymbol{v}^* | t) f(\boldsymbol{x}, \boldsymbol{v}_1^* | t) - \\ & - f(\boldsymbol{x}, \boldsymbol{v} | t) f(\boldsymbol{x}, \boldsymbol{v}_1 | t)] \, \sigma(\|\boldsymbol{v}_r\|, \hat{\boldsymbol{k}} \circ \boldsymbol{v}_r) \|\boldsymbol{v}_r\| d\boldsymbol{v}_1 d^2 \hat{\boldsymbol{k}} \end{aligned} \tag{2}$$

when written for a gas composed by a single monatomic species whose atoms have mass m. In Eqs. (1,2), $f(\boldsymbol{x}, \boldsymbol{v}|t)$ denotes the distribution function of atomic velocities $\boldsymbol{v}$ at spatial location $\boldsymbol{x}$ and time t, $\boldsymbol{F}(\boldsymbol{x}, \boldsymbol{v}|t)$ is an assigned external force field, whereas $\mathcal{C}(f, f)$ gives the collisional rate of change of f at the phase space point $(\boldsymbol{x}, \boldsymbol{v})$ at time t. As is clear from Eq. (2), $\mathcal{C}(f, f)$ is a non-linear functional of f, whose precise structure depends on the assumed atomic interaction forces through the differential cross section $\sigma(\|\boldsymbol{v}_r\|, \hat{\boldsymbol{k}} \circ \boldsymbol{v}_r)$. The dynamics of binary encounters determines σ as a function of the modulus $\|\boldsymbol{v}_r\|$ of the relative velocity $\boldsymbol{v}_r = \boldsymbol{v}_1 - \boldsymbol{v}$ of two colliding atoms and of the orientation of the unit impact vector $\hat{\boldsymbol{k}}$ with respect to $\boldsymbol{v}_r$ [7]. The collisional dynamics also determines the pre-collisional velocities $\boldsymbol{v}^*$ and $\boldsymbol{v}_1^*$ which are changed into $\boldsymbol{v}$ and $\boldsymbol{v}_1$ by a binary collision. For illustration purposes, it is worth mentioning that the collision integral $\mathcal{C}(f, f)$ simplifies to

$$\mathcal{C}(f,f) = \frac{d^2}{2} \int [f(\boldsymbol{x}, \boldsymbol{v}^*|t) f(\boldsymbol{x}, \boldsymbol{v}_1^*|t) - f(\boldsymbol{x}, \boldsymbol{v}|t) f(\boldsymbol{x}, \boldsymbol{v}_1|t)] \, |\hat{\boldsymbol{k}} \circ \boldsymbol{v}_r| d\boldsymbol{v}_1 d^2 \hat{\boldsymbol{k}} \tag{3}$$

for a dilute gas of hard spheres of diameter d. In this case $\boldsymbol{v}^*$ and $\boldsymbol{v}_1^*$ are obtained from $\boldsymbol{v}$, $\boldsymbol{v}_1$ and $\hat{\boldsymbol{k}}$ by the simple relationships

$$\boldsymbol{v}^* = \boldsymbol{v} + (\boldsymbol{v}_r \circ \hat{\boldsymbol{k}}) \hat{\boldsymbol{k}} \tag{4}$$

$$\boldsymbol{v}_1^* = \boldsymbol{v}_1 - (\boldsymbol{v}_r \circ \hat{\boldsymbol{k}}) \hat{\boldsymbol{k}} \tag{5}$$

Obtaining numerical solutions of the Boltzmann equation for realistic flow conditions is a challenging task because the unknown function depends, in principle, on seven variables. Moreover, the computation of $\mathcal{C}(f, f)$ requires the approximate evaluation of a fivefold integral. Numerical methods for rarefied gas dynamics studies can be roughly divided into three groups:

(a) Particle methods
(b) Semi-regular methods
(c) Regular methods

Methods in group (a) originate from the Direct Simulation Monte Carlo (DSMC) scheme proposed by G.A. Bird [8]. They are by far the most popular and widely used simulation methods in rarefied gas dynamics. The distribution

function is represented by a number of mathematical particles which move in the computational domain and collide according to stochastic rules derived from Eqs. (1,2). Macroscopic flow properties are usually obtained by time averaging particle properties. If the averaging time is long enough, then accurate flow simulations can be obtained by a relatively small number of particles. The method can be easily extended to deal with mixtures of chemically reacting polyatomic species [8] and to dense fluids [9]. Although DSMC (in its traditional implementation) is to be recommended in simulating most of rarefied gas flows, it is not well suited to the simulation of low Mach number or unsteady flows. Attempts have been made to extend DSMC in order to improve its capability to capture the small deviations from the equilibrium condition met in low Mach number flows [10,11]. However, in simulating high frequency unsteady flows, typical of microfluidics application to MEMS [4], the possibility of time averaging is lost or reduced. Acceptable accuracy can then be achieved by increasing the number of simulation particles or superposing several flow snapshots obtained from statistically independent simulations of the same flow; in both cases the computing effort is considerably increased.
Methods in groups (b) and (c) adopt similar strategies in discretizing the distribution function on a regular grid in the phase space and in using finite difference schemes to approximate the streaming term on the l.h.s of Eq. (1). However, they differ in the way the collision integral is evaluated. In semi-regular methods $\mathcal{C}(f, f)$ is computed by Monte Carlo or quasi Monte Carlo quadrature methods [12,13] whereas deterministic integration schemes are used in regular methods [14]. Whatever method is chosen to compute the collision term, the adoption of a grid in the phase space considerably limits the applicability of methods (b) and (c) to problems where particular symmetries reduce the number of spatial and velocity variables. As a matter of fact, a spatially three-dimensional problem would require a memory demanding six-dimensional phase space grid. Extensions to polyatomic gases are possible [15] but the necessity to store additional variables associated with internal degrees of freedom further limits the applications to multi-dimensional flows. In spite of the drawbacks listed above, the direct solution of the Boltzmann equation by semi-regular or regular methods is a valid alternative to particle schemes in studying unsteady or low speed flows. Actually, when the deviation from equilibrium is small a limited number of grid points in the velocity space is sufficient to provide accurate and noise free approximations of f, therefore simulations of multi-dimensional flows are feasible on modern personal computers in a wide range of Knudsen numbers [16].
An important feature of kinetic equations for dilute gases is the locality of the collision term; the collisional rate of change $\mathcal{C}(f, f)$ at the spatial location $\boldsymbol{x}$ is completely determined by $f(\boldsymbol{x}, \boldsymbol{v}|t)$. Hence, the time consuming evaluation of the collision integral can be concurrently executed at each spatial grid point on parallel computers. As shown below, the numerical algorithm associated with regular or semi-regular methods is ideally suited for the parallel architecture provided by commercially available GPUs. The aim of the paper is

to describe an efficient algorithm specifically tailored for solving kinetic equations onto GPUs using CUDA™ programming model [17]. The efficiency of the algorithm is assessed by solving the classical one-dimensional shock wave structure and a low speed two-dimensional driven cavity flow. It is shown that it is possible to cut the computing time of the sequential codes of two order of magnitudes by a proper reformulation of the algorithm to be executed on a GPU.
In order to make the algorithm development easier, the computations presented here have been performed by replacing $\mathcal{C}(f,f)$ with its simpler BGKW approximation [6]. This choice eliminates the intricacies connected with the numerical evaluation of the Boltzmann collision integral and allows easier identification of bottlenecks and optimization strategies. As will be shown in a separate paper, the full Boltzmann equation can be solved within the same general algorithmic framework by adopting a Monte Carlo quadrature method.
This paper is organized as follows. Section II is devoted to a concise description of the mathematical model and the adopted numerical method. In Section III the key aspects of the GPU hardware architecture and CUDA™ programming language are briefly described. Sections IV and V are devoted to the description of the test problems and the discussion of the results. Concluding remarks are presented in Section VI.

2 Theoretical and numerical background

Both from the theoretical and computational point of view, it is often convenient to replace the full Boltzmann equation with a model equation having a simplified collision term. In the kinetic model proposed by Bhatnagar Gross and Krook [18] and independently by Welander [19], $\mathcal{C}(f,f)$ is replaced by the expression $\nu\,(\Phi - f)$. Accordingly, Eq. (1) is turned into the following kinetic equation:

$$\frac{\partial f}{\partial t} + \boldsymbol{v} \circ \nabla_{\boldsymbol{x}} f + \frac{1}{m} \nabla_{\boldsymbol{v}} \circ (\boldsymbol{F} f) = \nu\,(\Phi - f) \tag{6}$$

In Eq. (6) ν is the collision frequency, whereas Φ is the local equilibrium Maxwellian distribution function given by the expression

$$\Phi(\boldsymbol{x}, \boldsymbol{v}|t) = \frac{n(\boldsymbol{x}|t)}{\left[2\pi R T(\boldsymbol{x}|t)\right]^{3/2}} \exp\left\{-\frac{[\boldsymbol{v} - \boldsymbol{V}(\boldsymbol{x}|t)]^2}{2RT(\boldsymbol{x}|t)}\right\} \tag{7}$$

If ν does not depend on the velocity $\boldsymbol{v}$, then conservation of mass, momentum and energy requires that n, $\boldsymbol{V}$ and T in Eq. (7) coincide with the local values of density, bulk velocity and temperature obtained from f by the relationships

$$n = \int f d\boldsymbol{v} \qquad \mathbf{V} = \frac{1}{n}\int f \boldsymbol{v} d\boldsymbol{v} \qquad T = \frac{1}{3Rn}\int f(\boldsymbol{v} - \mathbf{V})^2 d\boldsymbol{v} \tag{8}$$

being R the specific gas constant. The above expressions show that Eq. (6) is a strongly non-linear integro-differential equation, in spite of the linear appearance of its r.h.s..
As is well known, the BGKW model predicts an incorrect value of the Prandtl number in the hydrodynamic limit [6]. Hence, ν can be adjusted to obtain either the correct viscosity or heat conductivity, but not both. If viscosity is selected, then ν is given by the following expression:

$$\nu = \frac{nRT}{\mu} \tag{9}$$

being $\mu(T)$ the gas viscosity.

2.1 Outline of the numerical method

In view of the exploratory nature of the present work, Eq. (6) has been solved by a simple numerical method which will be illustrated on a spatially one-dimensional problem. The extension to two or three-dimensional geometries is straightforward.
In absence of external forces and in one-dimensional slab geometry Eq. (6) takes the form:

$$\frac{\partial f}{\partial t} + v_x \frac{\partial f}{\partial x} = \nu \left(\Phi - f \right) \tag{10}$$

where x is the single spatial coordinate and v_x the x-component of the velocity vector $\boldsymbol{v} = (v_x, v_y, v_z)$. The spatial domain is a finite interval of the real axis, divided into N_x cells of equal size Δx. The infinite three-dimensional velocity space is replaced by a rectangular box divided into $N_v = N_{v_x} \times N_{v_y} \times N_{v_z}$ cells of equal volume $\Delta\mathcal{V}$, N_{v_α} being the number of velocity nodes associated with the velocity component v_α. The size and position of the "velocity box" in the velocity space have to be properly chosen, in order to contain the significant part of f at any spatial position. The distribution function is assumed to be constant within each cell of the phase space. Hence, f is represented by the array $f_{i,\boldsymbol{j}}(t) = f(x(i), v_x(j_x), v_y(j_y), v_z(j_z)|t)$, being $x(i), v_x(j_x), v_y(j_y), v_z(j_z)$ the values of the spatial coordinate and velocity components in the center of the phase space cell $(i, \mathbf{j})$ and $\boldsymbol{j} = (j_x, j_y, j_z)$.
The algorithm that advances $f_{i,\boldsymbol{j}}(t)$ to $f_{i,\boldsymbol{j}}(t + \Delta t)$ is constructed by time-splitting the evolution operator into a free streaming step, in which the r.h.s. of Eq. (6) is neglected, and a purely collisional step, in which spatial motion is frozen and only the effect of the r.h.s. are taken into account. More precisely, the distribution function $f^n_{i,\boldsymbol{j}} = f_{i,\boldsymbol{j}}(t_n)$ at time level t_n is advanced to its value $f^{n+1}_{i,\boldsymbol{j}} = f_{i,\boldsymbol{j}}(t_{n+1})$ at time level $t_{n+1} = t_n + \Delta t$ by computing an intermediate value $\tilde{f}^{n+1}_{i,\boldsymbol{j}}$ from the free streaming equation

$$\frac{\partial f}{\partial t} + v_x \frac{\partial f}{\partial x} = 0 \tag{11}$$

Eq. (11) is solved by a simple first order upwind scheme

$$\tilde{f}_{i,\boldsymbol{j}}^{n+1} = \begin{cases} \left(1 - \dfrac{v_x(j_x)\Delta t}{\Delta x}\right) f_{i,\boldsymbol{j}}^n + \dfrac{v_x(j_x)\Delta t}{\Delta x} f_{i-1,\boldsymbol{j}}^n & v_x(j_x) > 0 \\ \left(1 + \dfrac{v_x(j_x)\Delta t}{\Delta x}\right) f_{i,\boldsymbol{j}}^n - \dfrac{v_x(j_x)\Delta t}{\Delta x} f_{i+1,\boldsymbol{j}}^n & v_x(j_x) < 0 \end{cases} \tag{12}$$

After completing the free streaming step, macroscopic variables n_i, $\boldsymbol{V}_i$ and T_i are computed at each spatial grid point and $f_{i,\boldsymbol{j}}^{n+1}$ is finally obtained by solving the homogeneous relaxation equation

$$\frac{\partial f}{\partial t} = \nu(\Phi - f) \tag{13}$$

Since n, $\boldsymbol{V}$ and T are conserved during homogeneous relaxation, Eq. (13) can be exactly solved to obtain

$$f_{i,\boldsymbol{j}}^{n+1} = [1 - \exp(-\nu_i \Delta t)]\, \Phi_{i,\boldsymbol{j}} + \exp(-\nu_i \Delta t) \tilde{f}_{i,\boldsymbol{j}}^{n+1} \tag{14}$$

in each cell $(i, \boldsymbol{j})$ of the phase space. The time step Δt has been set equal to a fraction of $1/\overline{\nu}$, being $\overline{\nu}$ a constant such that inequality $\nu_i \leq \overline{\nu}$ holds at each spatial cell. Such limitation on Δt ensures good accuracy but could lead to violation of the stability condition of the upwind scheme used in the free streaming sub-step. To overcame this difficulty, we note that the exact solution of the streaming term is

$$f(x, \mathbf{v}, t + \Delta t) = f(x - v_x \Delta t, \mathbf{v}, t) \tag{15}$$

Thus, for each molecular velocity, $v_x(j_x)$, the value of the distribution function in the cell $(i, \mathbf{j})$ of the phase space can be obtained by first translating the distribution function by a number of cells equal to the integer part of the Courant number $C = v_x(j_x)\Delta t/\Delta x$, $[C]$, and then applying expressions (12) for the residual time step advancement. It should be observed that the density, bulk velocity and temperature obtained from the discretized Maxwellian distribution function $\Phi_{i,\boldsymbol{j}}$ are not exactly equal to n_i, $\boldsymbol{V}_i$ and T_i. To ensure exact conservation of mass momentum and energy, the discretized $\Phi_{i,\boldsymbol{j}}$ should be computed from Eq. (7) by using effective values $\tilde{n}_i$, $\tilde{\boldsymbol{V}}_i$ and $\tilde{T}_i$ which are obtained by requiring that the moments of the *discretized* Maxwellian coincide with n_i, $\boldsymbol{V}_i$ and T_i [12]. The adoption of the correction method is not always necessary, since mass, momentum and energy errors are very small, even for coarse velocity space grids. Numerical tests have shown that calculating effective local values $\tilde{n}_i$, $\tilde{\boldsymbol{V}}_i$ and $\tilde{T}_i$ to force exact conservation of collisional invariants did not affect appreciably the solutions of the problems described below.

As is clear, both the free streaming and the relaxation sub-steps can be easily parallelized, each of them consisting of a number of independent threads.

3 GPU and CUDA™ overview

NVIDIA® GPU is built around a fully programmable processors array organized into a number of multiprocessors with a SIMD-like architecture [17], i.e. at any given clock cycle, each core of the multiprocessor executes the same instruction but operates on different data. CUDA™ is the high level programming language specifically created for developing applications on this platform.
A CUDA™ program is organized into a serial program which runs on the host CPU and one or more kernels which define the computation to be performed in parallel by a massive number of threads. Threads are organized into a three-level hierarchy. At the highest level, all threads form a grid; they all execute the same kernel function. Each grid consists of many different blocks which contain the same number of threads. A single multiprocessor can manage a number of blocks concurrently up to the resource limits. Blocks are independent, meaning that a kernel must execute correctly no matter the order in which blocks are run. A multiprocessor executes a group of threads beloging to the active block, called warp. All threads of a warp execute the same instruction but operate on different data. If a kernel contains a branch and threads of the same warp follow different paths, then the different paths are executed sequentially (warp divergence) and the total run time is the sum of all the branches. Divergence and reconvergence are managed in hardware but may have a serious impact on performance. When the instruction has been executed, the multiprocessor moves to another warp. In this manner the execution of threads is interleaved rather than simultaneous.
Each multiprocessor has a number of registers which are dynamically partitioned among the threads running on it. Registers are memory spaces that are readable and writable only by the thread to which they are assigned. Threads of a single block are allowed to syncronize with each other and are available to share data through a high-speed shared memory. Threads from different blocks in the same grid may coordinate only via operations in a slower global memory space which is readable and writeable by all threads in a kernel as well as by the host. Shared memory can be accessed by threads within a block as quickly as accessing registers. On the contrary, I/O operations involving global memory are particularly expensive, unless access is coalesced [17]. Because of the interleaved warp execution, memory access latency are partially hidden, i.e., threads which have read their data can be performing computations while other warps running on the same multiprocessor are waiting for their data to come in from global memory. Note, however, that GPU global memory is still ten time faster than the main memory of recent CPUs.
Code optimization is a delicate task. In general, applications which require many arithmetic operations between memory read/write, and which minimize the number of out-of-order memory access, tend to perform better. Number of blocks and number of threads per block have to be chosen carefully. There

should be at least as many blocks as multiprocessors in the device. Running only one block per multiprocessor can force the multiprocessor to idle during thread synchronization and device memory reads. By increasing the number of blocks, on the other hand, the amount of available shared memory for each block diminuishes. Allocating more threads per block is better for efficient time slicing, but the more threads per block, the fewer registers are available per thread.

4 Shock wave

4.1 Formulation of the problem

The propagation of a planar shock wave is a classical application of kinetic equations which is a rather natural choice as a benchmark problem because of the considerable number of previous studies [20,21]. In the wave front reference frame, the stationary flow field is assumed to be governed by the one-dimensional steady BGKW equation

$$v_x \frac{\partial f}{\partial x} = \nu(\Phi - f) \tag{16}$$

x being the spatial coordinate which spans the direction normal to the (planar) wave front. It is further assumed that, far from the wave front, the distribution function $f(x, \boldsymbol{v})$ satisfies the boundary conditions

$$\lim_{x \to \mp\infty} f(x, \boldsymbol{v}) = \Phi^{\mp}(\boldsymbol{v}) = \frac{n^{\mp}}{(2\pi R T^{\mp})^{3/2}} \exp\left[-\frac{(v_x - V^{\mp})^2 + v_y^2 + v_z^2}{2RT^{\mp}}\right] \tag{17}$$

where $n^{\mp}$, $V^{\mp}$ and $T^{\mp}$ are the upstream and downstream values of number density, velocity and temperature, respectively. The parameters of the equilibrium states specified by Eq. (17) are connected by the Rankine-Hugoniot relationships

$$\frac{V^-}{V^+} = \frac{n^+}{n^-} = \frac{4(M^-)^2}{(M^-)^2 + 3} \qquad \frac{T^+}{T^-} = \frac{[5(M^-)^2 - 1]\,[(M^-)^2 + 3]}{16(M^-)^2} \tag{18}$$

In Eqs. (18) M^- denotes the upstream infinity Mach number defined as

$$M^- = \frac{V^-}{(\gamma R T^-)^{1/2}} \tag{19}$$

being $\gamma = 5/3$ the specific heat ratio of a monatomic gas.
The numerical scheme described in section 2.1 has been adopted to obtain approximate solutions of Eq. (16) with boundary conditions (17) as long time

limit of solutions of Eq. (10) with identical boundary conditions and initial condition

$$f(x,\boldsymbol{v}|0) = \begin{cases} \Phi^{-}(\boldsymbol{v}) & x < 0 \\ \Phi^{+}(\boldsymbol{v}) & x > 0 \end{cases} \tag{20}$$

The computations reported have been carried out for both a weak, $M^{-} = 1.5$, and a medium, $M^{-} = 3.0$, shock wave. The collision frequency has been obtained from Eq. (9), assuming that the viscosity is given by the expression

$$\mu(T) = \mu_0 \left(\frac{T}{T_0}\right)^{0.74} \tag{21}$$

In Eq. (21), T_0 is a reference temperature, μ_0 is the value of the viscosity at the reference temperature. The temperature exponent has been set equal to 0.74 to match the computational conditions of Ref. [20] whose results have been used to asses the accuracy of the calculations presented here. An adimensional form of the Eq. (10) has been adopted in actual computations by normalizing velocity $\boldsymbol{v}$ to $\sqrt{2RT^{-}}$, time t to $\tau^{-} = 1/\nu^{-}$ and spatial coordinate x to the mean free path $\lambda^{-} = \sqrt{2RT^{-}}\tau^{-}$. The reference value ν^{-} for the collision frequency has been obtained by setting $T_0 = T^{-}$ in Eq. (21). The infinite physical space has been replaced by the finite interval $[-L/2, L/2]$ which has been divided into N_x identical cells of width $\Delta x = L/N_x$. The adimensional size L of the spatial domain has been set equal to 70, varying N_x between 128 and 18432. The cell number N_x has been increased well above the limit imposed by accuracy in order to investigate the GPU performances as a function of computational load. Similarly, the velocity space has been replaced by a parallelepiped in which each velocity component v_α varies in a finite interval, divided into N_{v_α} equal cells. The position of parallelepiped in the velocity space and the cell number vary with the chosen Mach number. In case of the weak shock wave, $M^{-} = 1.5$, the same number of grid points has been used for the three normalized velocity components by setting $N_{v_\alpha} = 16$, with $v_x \in [-5, 7]$ and $v_y, v_z \in [-6, 6]$. In case of the $M^{-}_{\infty} = 3.0$ shock wave, the grid point setting has been changed to $N_{v_\alpha} = 30$, with $v_x \in [-10, 12]$ and $v_y, v_z \in [-11, 11]$. Finally, the normalized time step Δt has been set equal to 0.05. Before describing the algorithm implementation and describing the results, it is worth observing that, for one and two-dimensional problems, the dimensionality of the velocity space associated with kinetic model equations having the structure of Eq. (6) can be accordingly reduced to one and two, respectively [22]. The reduction has not been made in the present work to keep the general structure of a three-dimensional code and, as mentioned above, to investigate the hardware response to heavy computer storage demand.

4.2 CUDA™ implementation

The code to numerically solve Eq. (16) is organized into a host program, which deals with all memory management and other setup tasks, and two kernels running on the GPU. One performs the streaming step and the other one performs the collision step and compute the macroscopic quantities as well. Alghorithms 1 and 2 list the pseudocodes of both kernels. For clarity of presentation, the pseudocode of the streaming step refers to the case of $v_x > 0$. Because of their different impact on the code performance, we distinguish the slow global memory reads, $\Leftarrow$, and writes, $\Rightarrow$, from the fast reads, $\leftarrow$, and writes, $\rightarrow$, from local registers and shared memory.

As shown by Eq. (15), for each given cell of the velocity space, the streaming step involves the distribution function evaluated at different space locations. The key performance enhancing strategy is to allow threads to cooperate in the shared memory. The threads should thus be grouped into as many blocks as the cells in the velocity space with a number of threads per block equals to the number of cells in the physical space. In practical applications, however, the number of cells in the physical space is greater than the maximum allowable number of threads per block. In order to fit into the device's resources, hence, the number of threads per block, N_t, is set to a lower value which is chosen to maximize the utilization of registers and shared memory usage. When a block become active, each thread loads one element of the distribution function from global memory, stores it into shared memory (line 7) and then update its value according to Eqs. (12) (line 13). This procedure is then repeated sequentially N_x/N_t times. To ensure non-overlapping access, threads are synchronized at the onset of both reading from and writing to the global memory (lines 12 and 15). In order the access to the global memory to be coalesced, the discretized distribution function has been organized such that the value which refers to cells which are adjacent in the physical space are stored in contiguous memory locations. A random memory access would determine otherwise a performance bottleneck. Threads which update boundary points perform calculations which are slightly different to account for the incoming Maxwellian flux from the boundary of the domain (line 5). This leads to a thread divergence which determines some code inefficiency. However, testing shows that the performance loss is small.

According to Eq. (14), in order to perform the relaxation step in a cell of the phase space, no information from nearby cells is needed. Hence the concurrent computation of the collision operator may be possible mapping each thread to a single cell in the phase space. This choice naturally fits for GPUs. In order to evaluate the local Maxwellian, Φ, however, one must first calculate in each cell of the physical space the macroscopic quantities, Eqs. (18). In the attempt of reducing data transfers from and to the global memory, the computation of the macroscopic quantities (lines 1-9) and the collision step (lines 11-13) are then performed in the same kernel, by having a thread associated to each cell of the physical space. Although this choice reduce the overall number of threads, it is not quite limiting since for realistic three-dimensional problems, one would probably refine the physical grid more than the velocity grid.

4.3 Results and discussion

In this section, we first validate the code by solving the plane shock structure problem and then we evaluate its performance by comparing the GPU and CPU execution times. We chose representative commercial products from both the CPU and GPU markets: Intel® Core™ Duo Quad Q9300 running at 2.50 GHz with 6 MB of L2 cache and with 4 GB of main memory, and an NVIDIA® GeForce GTX 260 with CUDA™ version 2.0. The GTX 260 consists of 24 streaming multiprocessors. Each multiprocessor has 8 streaming processors for a total of 192 units, clocked at 1.24 GHz. Each group of streaming processors shares one 16 kB of fast per-block shared memory while the GPU has 896 MB of device memory.

Figures 1a and 1b show the velocity and temperature profiles versus the x coordinate. Solid and dashed lines are the results from the numerical solution of Eq. (16) for $M^- = 1.5$ and $M^- = 3$, respectively. Solid circles and squares are the results presented in Ref. [20] for $M^- = 1.5$ and $M^- = 3$, respectively. The agreement is good and provides a validation of the numerical code.

The performance of the GPU implementation is compared against the single-threaded version running on the CPU by computing the speedup factor $S = T_{\text{CPU}}/T_{\text{GPU}}$, where T_{CPU} and T_{GPU} are the times used by the CPU and GPU to process at each time step one element of the discretized distribution function, respectively. We first examine the performance of each kernel and then analyze the overall speedup of the program. Times are measured after initial setup, e.g., after file I/O, and do not include the time required to transfer data between the disjoint CPU and GPU memory spaces.

Figure 2 shows the speedup of the streaming kernel, S_{s}, versus the number of cells in the physical space. Solid line with circles and dashed line with squares are the results for a different number of cells in the velocity space, $N_{v_\alpha} = 16$ and $N_{v_\alpha} = 30$, respectively. In both cases, the speedup sharply increases and then level off at about $N_x \backsimeq 3000$, where the GPU capability is fully exploited. For a greater number of cells in the physical space, the streaming step scales linearly with the elements of the discretized distribution function. The speedup decreases with the number of cells in the velocity space but, even in the worst case, it is still about 200.

Figure 3 shows the same as Fig. 2 but for the collision kernel, Unlike the streaming kernel, the speedup of the collision kernel, S_{c}, is greatly improved by increasing the number of cells in the velocity space.
Figures 4a and 4b show the relative time, spent to perform the streaming kernel, T_{s}, and the collision kernel, T_{c}, versus the number of cells in the physical space, for $M^- = 1.5$ and $M^- = 3$, respectively. As expected, the collision is much more time consuming than the streaming kernel. Moreover, the relative time does not appear to depend strongly on the number of cells in the velocity space.
Figure 5 shows the overall speedup, S_{t}, for the two test cases. Notation is the same as Figs. 2 and 3. The overall speedup for each case is in between S_{s} and S_{c}, which is not unexpected. In fact, S_{t} is the weighted average of the streaming and collision speedups with weight the relative time T_i spent by the GPU to execute each kernel, i.e., $S_{\text{t}} = T_{\text{s}}S_{\text{s}} + T_{\text{c}}S_{\text{c}}$.

5 Driven cavity

5.1 Formulation of the problem

The driven cavity flow is a classical multidimensional benchmark problem since, in spite of its simple geometry, it contains most of the features which appear in more complicated problems described by kinetic equations. A gas is confined in a two-dimensional square cavity and the flow is driven by a uniform translation of the top with velocity $U_{\text{W}}\mathbf{e}_x$. The gas flow is supposed to be governed by the two-dimensional steady BGKW equation

$$v_x \frac{\partial f}{\partial x} + v_y \frac{\partial f}{\partial y} = \nu(\Phi - f) \tag{22}$$

It is further assumed that all the walls are isothermal and that the particles which strike the walls are re-emitted according to the Maxwell's scattering kernel with complete accommodation

$$f(\boldsymbol{x}, \boldsymbol{v}) = \Phi_{\text{W}}(\mathbf{v}) = \frac{n_{\text{W}}}{(2\pi RT_0)^{1/2}} \exp\left[-\frac{(\boldsymbol{v} - \boldsymbol{V}_{\text{W}})^2}{2RT_0}\right], \quad (\boldsymbol{v} - \boldsymbol{V}_{\text{W}}) \circ \boldsymbol{n} > 0 \tag{23}$$

where $\boldsymbol{x}$ is a point of the boundary, $\boldsymbol{n}$ the inward normal, $\mathbf{V}_{\text{W}}$ the wall velocity and n_{W} the wall density defined as

$$n_{\text{W}} = \left(\frac{2\pi}{RT_0}\right)^{1/2} \int_{(\boldsymbol{v} - \boldsymbol{V}_{\text{W}}) \circ \boldsymbol{n} < 0} |(\boldsymbol{v} - \boldsymbol{V}_{\text{W}}) \circ \boldsymbol{n}| f \, d\boldsymbol{v} \tag{24}$$

in order to impose zero net mass flux at any boundary point.
The two-dimensional extension of the numerical scheme described in section 2.1 has been adopted to obtain approximate solutions of Eq. (22) with boundary conditions (23) as long time limit of the unsteady problem. The adimensional form of the governing equation has been obtained as described in section 4. In Ref. [16], the cavity flow problem has been solved by assuming that $V_\mathrm{W} \ll \sqrt{2RT_0}$ and thus Eq. (22) has been linearized around the equilibrium state at rest. In order to reproduce these results, the dimensionless lid velocity is here set to 0.01. The gas is thus in a weakly non-equilibrium state and the nonlinear results approach the linearized ones. The square cavity, $[0,\delta]\times[0,\delta]$, has been divided into $N_x = N_y = 160$ cells with uniform width. Here δ is the rarefaction parameter which is proportional to the inverse of the Knudsen number. The cavity flow problem has been solved over a wide range of the rarefaction parameter, $\delta \in [0.1 - 10]$. The computational grid in the physical space has been chosen to achieve the convergence of the results in the whole range of rarefaction parameter considered. The number of velocity cells have been set $N_{v_\alpha} = 20$ with $v_x, v_y, v_z \in [-3, 3]$. Finally, the time step has been varied in the range $10^{-4} - 10^{-2}$ depending on the rarefaction parameter.

5.2 Results and discussion

Figures 6a and 6b show the profiles of the horizontal component of the velocity, V_x/V_W, on the vertical plane crossing the center of a square cavity and the vertical component of the velocity, V_y/V_W, on the horizontal plane crossing the center of the top vortex, respectively, for two different value of the rarefaction parameter, $\delta = 0.1, 10$. Solid lines are the numerical results obtained by solving Eq. (22) with the parallel code, solid circles are the results reported in Ref. [16]. The agreement is quite good. The near linear profiles of the velocity in the central core of the cavity indicate the uniform vorticity region. In order to proceed with a more detailed comparison, we introduce two overall quantities, namely the mean dimensionless shear stress, D, along the moving plate and the dimensionless flow rate, G, of the main vortex. The former quantities is obtained by integrating the shear stress along the lid of the cavity, the latter by integrating the x-component of the velocity profile along the plane crossing the center of the cavity from the center of the top vortex up to the lid. Table 1 compares the prediction of D and G obtained by solving Eq. (22) with the parallel code and the values reported in Ref. [16], for different values of the rarefaction parameter. The agreement is good, the greatest mismatch being for the drag coefficient at $\delta = 10$. However the discrepancy is easily removed by increasing the number of cells in the physical space. Figure 7 shows the time spent for processing one cell of the the phase space at each time step, expressed in nanoseconds, versus the number of cells used to discretize the physical space, $N_r = N_x$ for the one-dimensional code and $N_r = N_x + N_y$ for the two-dimensional code. The solid and dashed lines are the results for the shock wave and driven cavity flow problems, respectively. The one-dimensional and two-dimensional codes fully utilize the GPU when the number of cells in the physical space is about 6000 and 9000, respectively. This difference is due to the fact that the one-dimensional code makes a better use of GPU's registers and shared memory. For a greater number of cells, the total execution time increases linearly and hence the time spent for processing one cell of the phase space at each time step is nearly constant. When the GPU is fully exploited, the two-dimensional code is slower than the one-dimensional code by a factor of about 2, which is not unexpected because of thread divergence determined by the greater number of boundary cells. Although the sequential version of the two-dimensional code has not been written, it can be safely inferred that the speedup of the two-dimensional code is about an half of the speedup of the one-dimensional code, on the basis of these results.

6 Conclusions

The aim of this paper is to describe the development of an algorithm to solve kinetic equations by exploiting the computing power of modern GPUs. Test gas flows have been studied by adopting the Bhatnagar-Gross-Krook-Welander (BGKW) kinetic model for the collision term in combination with a simple finite difference scheme. Numerical experiments with the one-dimensional shock wave structure problem and the two-dimensional driven cavity flow indicate that it is possible to cut down the computing time of the sequential codes up to two order of magnitudes. For instance, the solution of the two-dimensional unsteady driven cavity flow for $\delta = 10$ with $N_{v_\alpha} = 20$, $N_x = N_y = 160$ took about 7 minutes to execute 4100 time steps. It is worth to notice that if the specific two-dimensional nature of the problem is taken into account, then the dimensionality of the velocity space can be reduced by a standard projection procedure [22]. In this case, the computing time can be further reduced to about 40 seconds while keeping the same accuracy level. The algorithm described can easily be extended to three dimensions and to non-equilibrium flows involving mixtures, and/or chemical reactions [23]. Extension to polyatomic gas is possible as well provided that one adopts a proper representation of the distribution function to cope with the enlargement of the phase space due to internal degrees of freedom [15]. This paper describes the first stage in the development of algorithms for solving non-equilibrium gas flows onto GPUs. The extension of this algorithm to semi-regular method of solution to the full Boltzmann equation is presently being investigated and it will be considered in a future paper.

Acknowledgment

Support received from **Fondazione Cariplo** within the framework of project *"Fenomeni dissipativi e di rottura in micro e nano sistemi elettromeccanici"*, and **Galileo Programme** of Università Italo-Francese within the framework of project MONUMENT (MOdellizzazione NUmerica in MEms e NanoTecnologie) is gratefully acknowledged. The authors wish to thank Professor Dimitris Valougeorgis for providing his numerical results.

References

[1] E. Elsen, P. LeGresley, E. Darve, "Large calculation of the flow over a hypersonic vehicle using a GPU", J. Comp. Phys. 227 (2008) 10148-10161.

[2] G. Stantchev, W. Dorland, N. Gumerov, "Fast parallel Particle-To-Grid interpolation for plasma PIC Gsimulations on the GPU", J. Parallel Distrib. Comput. 68 (2008) 1339-1349.

[3] J. A. Anderson, C. D. Lorenz, A. Travesset, "General purpose molecular dynamics simulations fully implemented on graphics processing units", J. Comp. Phys. 227 (2008) 5342-5339.

[4] M. Gad-el-Hak, "The fluid mechanics of microdevices - the Freeman Scholar Lecture", J. Fluids Eng. (Trans. ASME) 121 (1999) 5-33.

[5] S. Lorenzani, L. Gibelli, A. Frezzotti, A. Frangi, C. Cercignani, "Kinetic approach to gas flows in microchannels", Nanoscale and Microscale Thermophysical Engineering 11 (2007) 211-226.

[6] C. Cercignani, The Boltzmann Equation and Its Applications, Springer-Verlag, New York, 1988.

[7] S. Chapman, T. G. Cowling, The mathematical theory of non-uniform gases, Cambridge University Press, 1990.

[8] G. A. Bird, Molecular Gas Dynamics and the Direct Simulation of Gas Flows, Oxford University Press, 1994.

[9] A. Frezzotti, L. Gibelli, S. Lorenzani, "Mean field kinetic theory description of evaporation of a fluid into vacuum", Phys. Fluids 17 (2005) 012102-12.

[10] T. M. M. Homolle, N. G. Hadjiconstantinou, "A low-variance deviational simulation Monte Carlo for the Boltzmann equation", J. Comput. Phys. 226 (2007) 2341-2358.

[11] W. Wagner, "Deviational particle Monte Carlo for the Boltzmann equation", Monte Carlo Methods and Applications 14 (2008) 191-268.

[12] A. Frezzotti, “Numerical study of the strong evaporation of a binary mixture”, Fluid Dynamics Research 8 (1991) 175-187.

[13] F. Tcheremissine, “Direct numerical solution of the Boltzmann Equation”, RGD24 AIP Conference proceeding 762 (2005) 677-685.

[14] V. V. Aristov, Direct Methods for Solving the Boltzmann Equation and Study of Nonequilibrium Flows, Springer-Verlag, New York, 2001.

[15] A. Frezzotti, “A numerical investigation of the steady evaporation of a polyatomic gas”, Eur. J. Mech. B: Fluids 26 (2007) 93-104.

[16] S. Vauritis, D. Valougeorgis, F. Sharipov, “Application of the integro-moment method to steady-state two-dimensional rerafeied gas flows subject to boundary induced discontinuities”, J. Comput. Phys. 227 (2008) 6272-6287.

[17] NVIDIA Corporation, “NVIDIA CUDA Programming Guide”, Jun. 2008. Version 2.0. http://www.nvidia.com/CUDA

[18] P. L. Bhatnagar, E. P. Gross, M. Krook, “A model for collision processes in gases. I. Small amplitude processes in charged and neutral one-component systems”, Phys. Rev. 94 (1954) 511-525.

[19] P. Welander, “On temperature jump in a rarefied gas”, Arkiv fiir Fysik 7 44 (1954) 507-553.

[20] H. W. Liepmann, R. Narasimha and M. T. Chahine, “Structure of a plane shock layer”, Phys. Fluids 5 (1962) 1313-1324.

[21] P. Kowalczyk, A. Palczewski, G. Russo, Z. Walenta, “Numerical solutions of the Boltzmann equation: comparison of different algorithms”, Eur. J. Mech. B: Fluids 27 (2008) 6274.

[22] C. K.Chu, “Kinetic-theoretic description of the formation of a shock wave”, Phys. Fluids 8 (1965) 12-22.

[23] M. Groppi, K. Aoki, G. Spiga, V. Tritsch, “Shock structure analysis in chemically reacting gas mixtures by a relaxation-time kinetic model”, Phys. Fluids 20 (2008) 117103-11.

Captions to Figures:

Fig. 1: (a) mean velocity and (b) temperature profiles versus the x coordinate. Solid and dashed lines are the results obtained with the parallel code for $M^- = 1.5$ and $M^- = 3$, respectively. Solid cirlces and squares are the results presented in Ref. [20] for $M^- = 1.5$ and $M^- = 3$, respectively.

Fig. 2: speedup of the streaming kernel, S_{s}, versus the number of cells in the physical space, N_x. Solid line with circles: $N_{v_\alpha} = 16$; dashed line with squares: $N_{v_\alpha} = 30$.

Fig. 3: speedup of the collision kernel, S_{c}, versus the number of cells in the physical space, N_x. Solid line with circles: $N_{v_\alpha} = 16$; dashed line with squares: $N_{v_\alpha} = 30$.

Fig. 4: relative time spent on the streaming and collision kernel for (a) $N_{v_\alpha} = 16$ and (b) $N_{v_\alpha} = 30$. Solid bar: streaming kernel; pattern bar: collision kernel.

Fig. 5: overall speedup, S_{tot}, versus the number of cells in the physical space, N_x. Solid line with circles: $N_{v_\alpha} = 16$; dashed line with squares: $N_{v_\alpha} = 30$.

Fig. 6: profiles of (a) the horizontal component of the velocity on the vertical plane crossing the center of the cavity and (b) the vertical component of the velocity on the horizontal plane crossing the center of the top vortex. Solid lines: numerical solution obtained with the parallel code; solid circles: solution reported in Ref. [16].

Fig. 7: time spent for processing one cell of the phase space versus the number cells used to discretized the physical space, N_r. Solid line: one-dimensional code. Dashed line: two-dimensional code. $N_{v_\alpha} = 16$.

Table 1: drag coefficient, D, and reduced flow rate, G, versus the rarefaction parameter, δ.

Algorithm 1 GPU pseudo-code of the streaming step

```
Require: $[C]$ is the integer part of the Courant number
Require: $\tilde{C}$ is the fractional part of the Courant number
Require: $N_x$ is the number of cells in the physical domain
Require: $N_t$ is the number of threads in each block
Require: $t = 0, \ldots N_t - 1$ is the index of the thread within the block
 1: $r \leftarrow N_x - N_t - [C]$
 2: $w \leftarrow N_x - N_t$
 3: for $i = 1$ to $N_x/N_t$ do
 4:     if $r + t < 0$ then
 5:         $f_{\text{sh}}(t+1) \leftarrow \Phi^-_{\mathbf{j}}$
 6:     else
 7:         $f_{\text{sh}}(t+1) \Leftarrow f^n_{r+t,\mathbf{j}}$
 8:     end if
 9:     if $t = 0$ then
10:         $f_{\text{sh}}(0) \Leftarrow f^n_{r-1,\mathbf{j}}$
11:     end if
12:     syncthreads
13:     $f_{\text{rg}} \leftarrow (1 - \tilde{C})\ f_{\text{sh}}(t+1) + \tilde{C}\ f_{\text{sh}}(t)$
14:     $f_{\text{rg}} \Rightarrow \tilde{f}^{n+1}_{w+t,\mathbf{j}}$
15:     syncthreads
16:     $r \leftarrow r - N_t$
17:     $w \leftarrow w - N_t$
18: end for
```

Algorithm 2 GPU pseudocode of the collision step

```
Require: $i$ is global index of the thread inside the grid
 1: for all $\mathbf{j}$ do
 2:     $f_{\mathrm{rg}} \Leftarrow \tilde{f}^{n+1}_{i,\mathbf{j}}$
 3:     $n_{\mathrm{rg}} \leftarrow n_{\mathrm{rg}} + f_{\mathrm{rg}}$
 4:     $\mathbf{V}_{\mathrm{rg}} \leftarrow \mathbf{V}_{\mathrm{rg}} + \mathbf{v}_{\mathbf{j}}\, f_{\mathrm{rg}}$
 5:     $e_{\mathrm{rg}} \leftarrow e_{\mathrm{rg}} + |\mathbf{v}_{\mathbf{j}}|^2\, f_{\mathrm{rg}}$
 6: end for
 7: $n_{\mathrm{rg}} \leftarrow n_{\mathrm{rg}}\, \Delta\mathcal{V}$
 8: $\mathbf{V}_{\mathrm{rg}} \leftarrow \mathbf{V}_{\mathrm{rg}}/n_{\mathrm{rg}}$
 9: $T_{\mathrm{rg}} \leftarrow (e_{\mathrm{rg}}/n_{\mathrm{rg}} - |\mathbf{V}_{\mathrm{rg}}|^2)/3$
10: for all $\mathbf{j}$ do
11:     $f_{\mathrm{rg}} \Leftarrow \tilde{f}^{n+1}_{i,\mathbf{j}}$
12:     $f_{\mathrm{rg}} \leftarrow [1 - \exp(-\nu_i \Delta t)]\, \Phi_{i,\boldsymbol{j}} + \exp(-\nu_i \Delta t) f_{\mathrm{rg}}$
13:     $f_{\mathrm{rg}} \Rightarrow f^{n+1}_{i,\mathbf{j}}$
14: end for
15: $n_{\mathrm{rg}} \Rightarrow n_i$
16: $\mathbf{V}_{\mathrm{rg}} \Rightarrow \mathbf{V}_i$
17: $T_{\mathrm{rg}} \Rightarrow T_i$
```

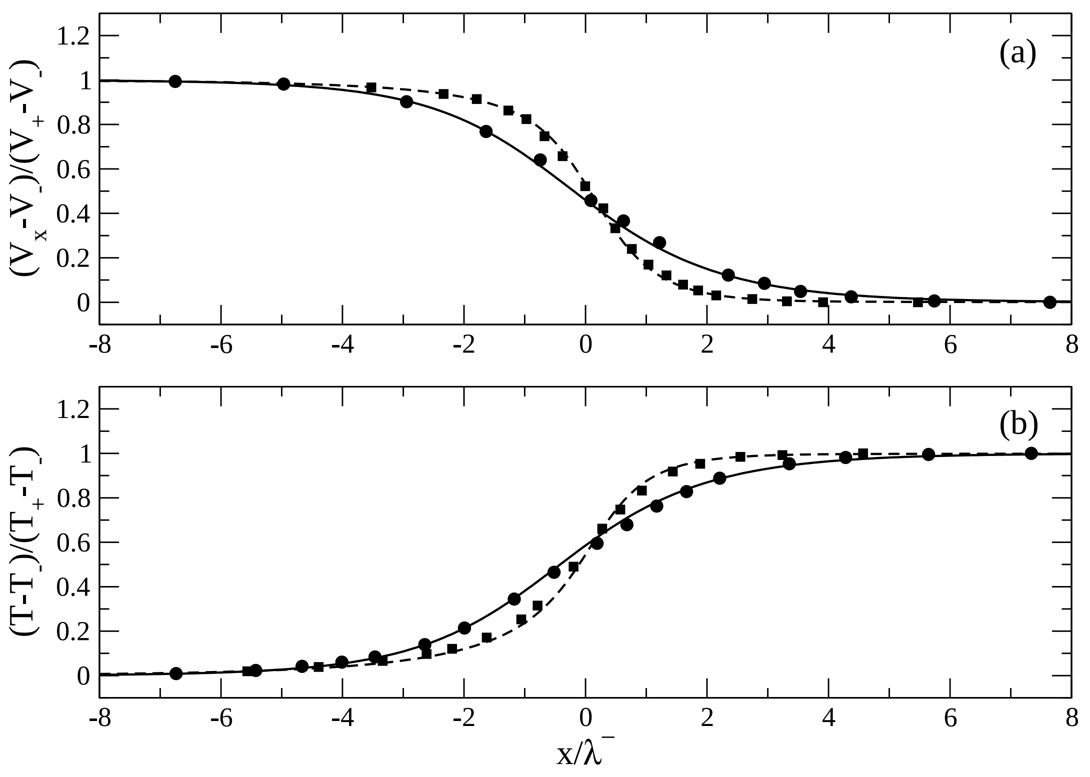

Fig. 1.

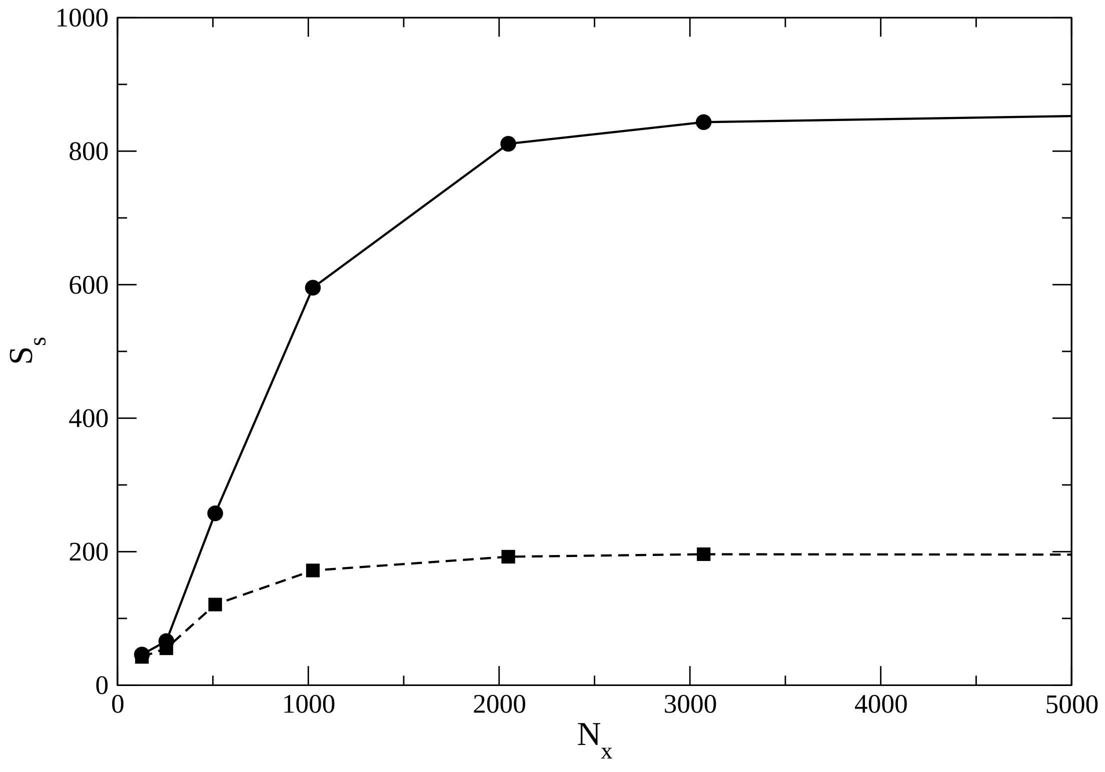

Fig. 2.

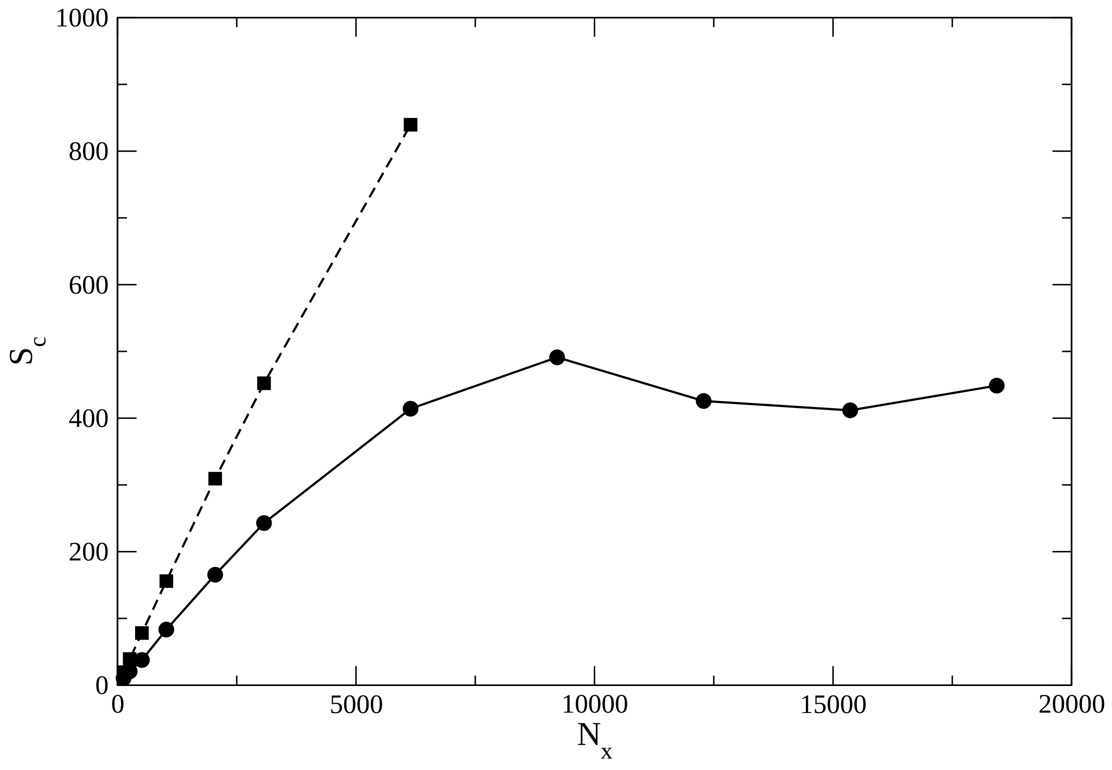

Fig. 3.

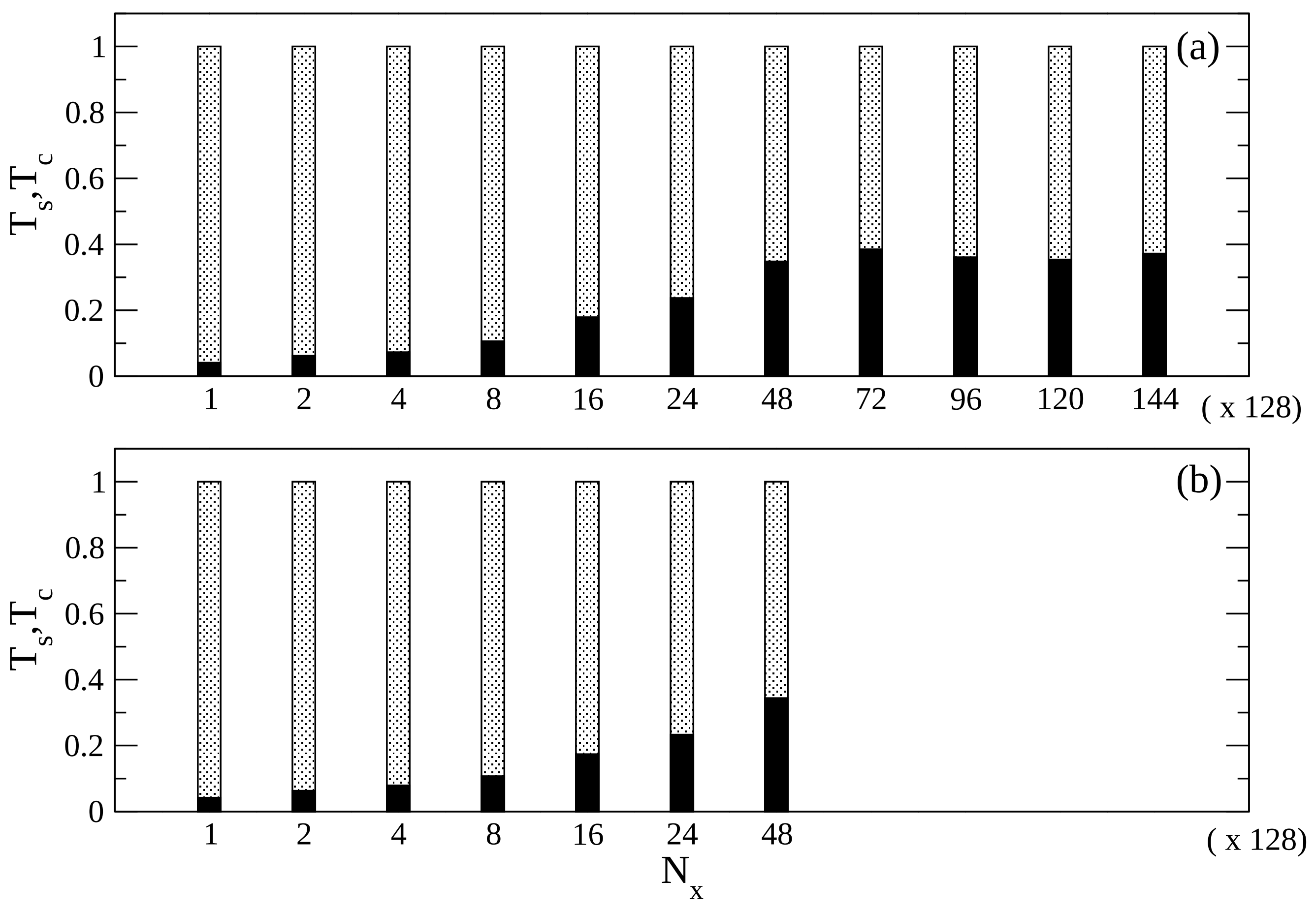

Fig. 4.

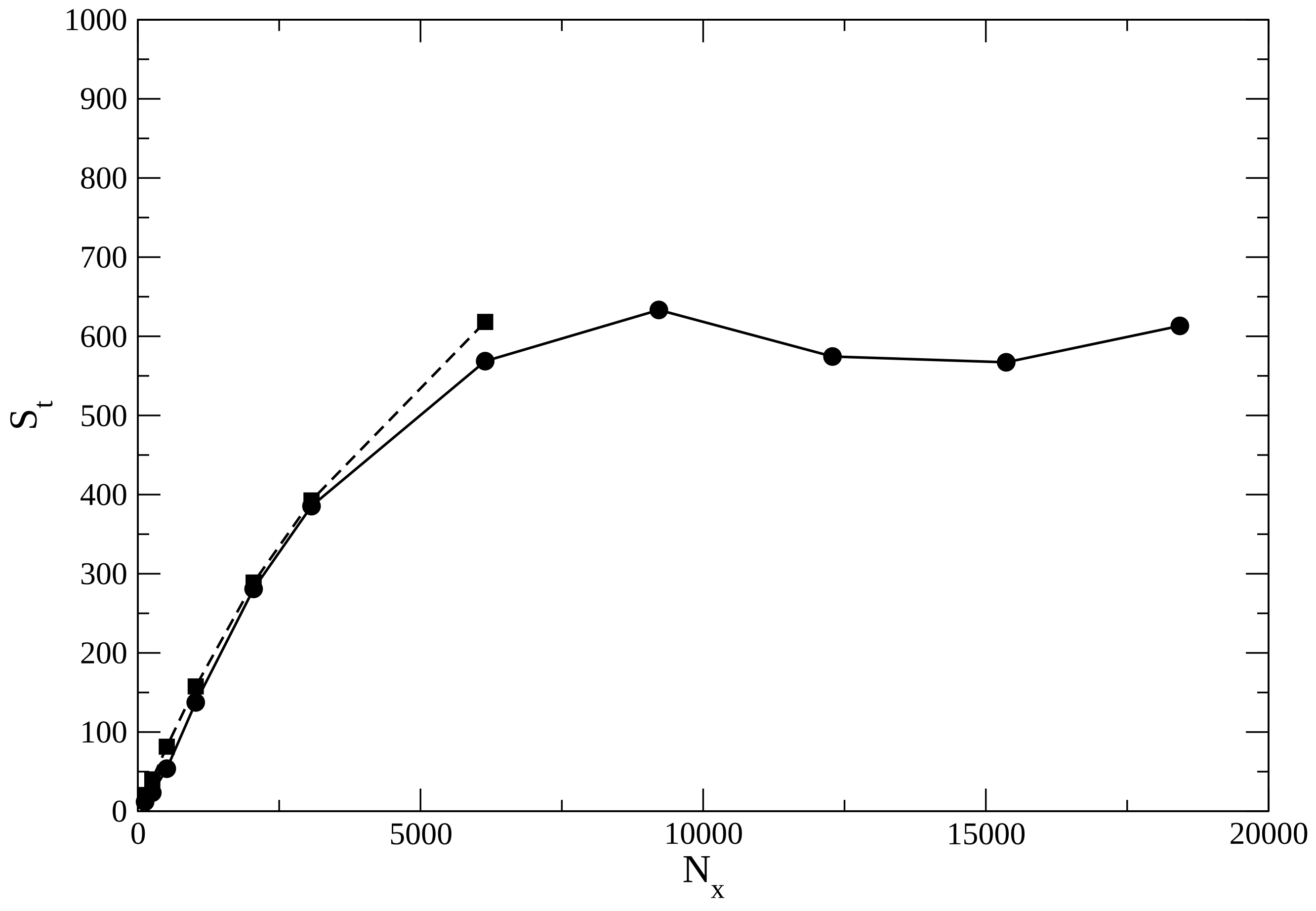
1000
900
800
700
600
500
400
300
200
100
0
S_t
0
5000
10000
15000
20000
N_x

Fig. 5.

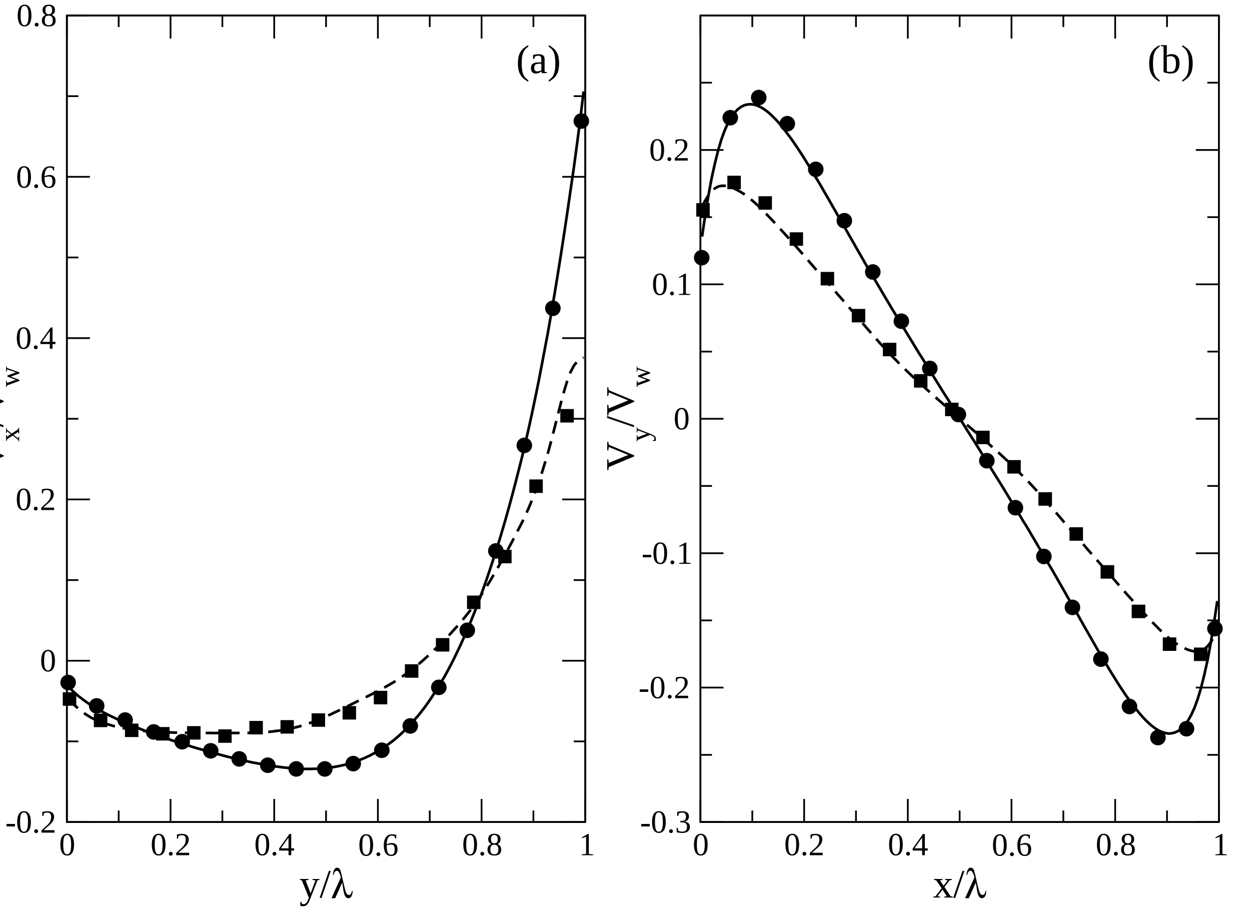

Fig. 6.

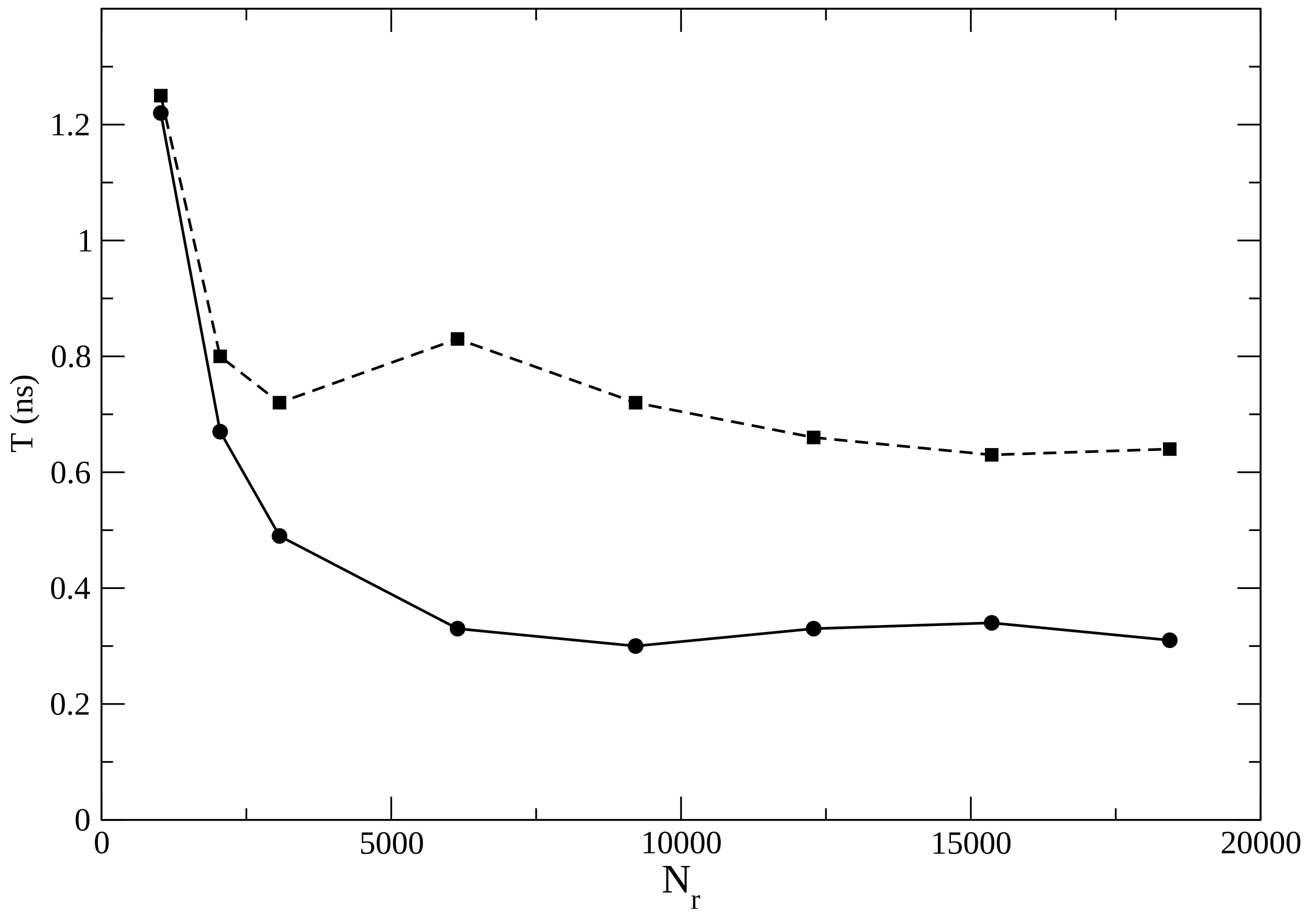

Fig. 7.

δ	D	D (Ref [16])	G	G (Ref [16])
0.1	0.675	0.678-0.676	0.0975	0.0973-0.0976
1	0.624	0.625-0.631	0.103	0.104-0.105
10	0.393	0.412-0.415	0.143	0.145-0.145

Table 1